Comparison of Interfaces and Abstract Classes

Interface	Abstract Class
Definition - A name, a set of constant definitions and a specification for a set of method signatures which is associated with that name.	Definition - An interface, a set of instance variable definitions and a partial implementation of the interface.
Inheritance Specifier *implements*	Inheritance Specifier *extends*
Multiple Inheritance	Single Inheritance
Is a subtype of class Object	Is a subtype of class Object

Comparison of Interfaces and Abstract Classes

Interface	Abstract Class
Used to define methods for implementation by multiple classes.	Used to define templates to be implemented by other classes.
Used to declare an object reference variable which can refer to any object of any class that implements the interface.	Used to declare an object reference variable which can refer to any object of any class that extends the abstract.
Enables polymorphic references in which the method being invoked is based on the particular object being referenced at the time	Enables polymorphic references in which the method being invoked is based on the particular object being referenced at the time

Comparison of Interfaces and Abstract Classes

Interface	Abstract Class
Used when inheritance of multiple different behaviors is required.	Used when there is a need to inherit instance variables as well as method definitions.
Or when a family of classes is required to implement a common behavior	Also supports application of one implementation of a method to instances of multiple classes.
Or when there is a desire to apply one implementation of a method to instances of multiple classes. (polymorphism)	

Comparison of Interfaces and Abstract Classes

Interface

```
interface Comparable
{  public  int
compareTo(Comparable c);
/* Returns:

 -1 if the invoking object instance is
lower in the sorting order  than the
argument instance ,

 0 if the two instances are equal

+1 if the invoking object instance is
higher in the sorting order than the
argument instance.
}
```

Abstract Class

```
abstract class Comparable
{  public abstract int
compareTo(Comparable c);
/* Returns:

 -1 if the invoking object instance is
lower in the sorting order  than the
argument instance ,

 0 if the two instances are equal

+1 if the invoking object instance is
higher in the sorting order than the
argument instance.
}
```

Example of Common Behavior: Philosopher and Dog

```
/* Definition of an Interface. */
interface Speaker
{
   public void speak ();
   public void announce (String str);
}
```



```
/* Implementation of an interface.*/
public class Dog implements Speaker
{ /* Prints this dog’s philosophy. */
   public void speak ()
   {
      System.out.println ("woof");
   }
*/ Prints this dog’s philosophy and the specified
       announcement.*/
   public void announce (String announcement)
   {
      System.out.println ("woof: " + announcement);
   }
}
```

Interfaces and Abstract Classes

```
/* Philosopher implements the Speaker interface. */
public class Philosopher implements Speaker
{  private String philosophy;
/*  Establishes this philosopher’s philosophy. */
   public Philosopher (String philosophy)
   {this.philosophy = philosophy;
   }
/*  Prints this philosophers’s philosophy.*/
   public void speak ()
   {System.out.println (philosophy);
   }
/* Prints the specified announcement. */
   public void announce (String announcement)
   {System.out.println (announcement);
   }
/* Prints this philosophers’s philosophy multiple times. */
   public void pontificate ()
   {for (int count=1; count <= 5; count++)
         System.out.println (philosophy);
   }
}
```


Interfaces and Abstract Classes

```
/* Demonstrates the use of an interface for polymorphic
references.*/
public class Talking
{
/* Instantiates two objects using an interface reference and
invokes one of the common methods. Then casts the
interface reference into a class reference to invoke its
unique method.*/
   public static void main (String[] args)
   {
      Speaker current;
      current = new Dog();
      current.speak();
      current = new Philosopher ("I think, therefore I am.");
      current.speak();
      ((Philosopher) current).pontificate();
   }
}
```

Interfaces and Abstract Classes

```
public abstract class Sortable
{  public abstract int compareTo(Sortable b);
}
public class ArrayAlg
{  public static void shellSort(Sortable[] a)
   {  int n = a.length;
      int incr = n / 2;
      while (incr >= 1)
      {  for (int i = incr; i < n; i++)
         {  Sortable temp = a[i];
            int j = i;
            while (j >= incr
               && temp.compareTo(a[j - incr]) < 0)
            {  a[j] = a[j - incr];
               j -= incr;
            }
            a[j] = temp;
         }
         incr /= 2;
      }
   }
}
```


Interfaces and Abstract Classes

```
interface Comparable
{  public  int compareTo(Comparable c); }

public class ArrayAlg
{  public static void shellSort(Comparable[] a)
   {  int n = a.length;
      int incr = n / 2;
      while (incr >= 1)
      {  for (int i = incr; i < n; i++)
         {  Comparable temp = a[i];
            int j = i;
            while (j >= incr
               && temp.compareTo(a[j - incr]) < 0)
            {  a[j] = a[j - incr];
               j -= incr;
            }
            a[j] = temp;
         }
         incr /= 2;
      }
   }
}
```

Interfaces and Abstract Classes

```
class Tile extends Rectangle implements Comparable
{  public Tile(int x, int y, int w, int h, int zz)
   {  super(x, y, w, h);
      z = zz;
   }
   public int compareTo(Object b)
   {  Tile tb = (Tile)b;
      return z - tb.z;
   }
   public String toString()
   {  return super.toString() + "[z=" + z + "]";
   }
   private int z;
}
```


Interfaces and Abstract Classes

```
import java.awt.*;
import java.util.*;
public class TileTest
{  public static void main(String[] args)
   {  Tile[] a = new Tile[20];
      int i;
      for (i = 0; i < a.length; i++)
         a[i] = new Tile(i, i, 10, 20,
            (int)(100 * Math.random()));
      ArrayAlg.shellSort(a);
      for (i = 0; i < a.length; i++)
         System.out.println(a[i]);
   }
}
```

Interfaces and Abstract Classes

```
public abstract class Message

{public Message(String from) { sender = from; }
  public abstract void play();
  public String getSender() { return sender; }

  private String sender;
/*  Requirement for instance variable means we cannot use an interface. */
}

public class TextMessage extends Message
{  public TextMessage(String from, String t)
  { super(from); text = t; }

  public void play() { System.out.println(text); }
 private String text;
}
```


Interfaces and Abstract Classes

```
public class VoiceMessage extends Message
{  public VoiceMessage(String from, String f)
  { super(from); filename = f; }

  public void play()
  {  try
    {  URL u = new URL("file", "localhost", filename);
      AudioClip clip = Applet.newAudioClip(u);
      clip.play();
    }
    catch(Exception e)
    {
      System.out.println("Can't open " + filename);
    }
  }

  private String filename;
}
```

Interfaces and Abstract Classes

```
public class Mailbox
{  public Message remove()
  {  if (nmsg == 0) return null;
     Message r = messages[out];
     nmsg--;
     out = (out + 1) % MAXMSG;
     return r;
  }

  public void insert(Message m)
  {  if (nmsg == MAXMSG) return;
     messages[in] = m;
     nmsg++;
     in = (in + 1) % MAXMSG;
  }
```


Interfaces and Abstract Classes

```
public String status()
  {  if (nmsg == 0) return "Mailbox empty";
     else if (nmsg == 1) return "1 message";
     else if (nmsg < MAXMSG) return nmsg + " messages";
     else return "Mailbox full";
  }

  private static final int MAXMSG = 10;
  private int in = 0;
  private int out = 0;
  private int nmsg = 0;
  private Message[] messages = new Message[MAXMSG];
}
```

Interfaces and Abstract Classes

```
import java.net.*;
import java.applet.*;
import corejava.*;
public class MailboxTest
{  public static void main(String[] args)
   {  Mailbox mbox = new Mailbox();
      while (true)
      {  System.out.println(mbox.status());
         String cmd = Console.readLine
               ("play, text, voice, quit> ");
         if (cmd.equals("play"))
         {  Message m = mbox.remove();
            if (m != null)
            {  System.out.println("From: " + m.getSender());
               m.play();
            }
         }
```


Interfaces and Abstract Classes

```
else if (cmd.equals("text"))
         {  String from = Console.readLine("Your name: ");
            boolean more = true;
            String msg = "";
            System.out.println
               ("Enter message, 'exit' when done");
            while (more)
            {  String line = Console.readLine();
               if (line.equals("exit"))
                  more = false;
               else msg = msg + line + "\n";
            }
            mbox.insert(new TextMessage(from, msg));
         }
         else if (cmd.equals("voice"))
         {  String from = Console.readLine("Your name: ");
            String msg
               = Console.readLine("Audio file name: ");
            mbox.insert(new VoiceMessage(from, msg));
         }
         else if (cmd.equals("quit"))
            System.exit(0);
      }
   }
}
```